32-应用的改进：如何改进我们的软件设计？

你好，我是郑晔！

前面两讲，我们分别讲了如何从头开始设计一个程序库和应用。但是在实际工作中，有很多时候，我们的工作并不是从头设计一个应用，而是改进一个既有项目的代码。既有项目的代码意味着什么呢？意味着各种问题。

我们一直在说，软件设计是一门关注长期变化的学问。越是在商业上成功的软件，存续的时间往往越长。存续的时间越长，往往就会有更多的麻烦。

我们先不说有些项目一开始就没有设计，一路混乱向前。即便是一个最初有着还算不错设计的项目，随着时间的积累、人员的更替、把前人的做法当作惯例等等事情的发生，项目的设计就会逐渐变得不堪重负。

我在前面的课程中也举过一些例子，虽然每人只改了一点点，最后却是积重难返。这就是一个项目缺乏设计守护的结果。好的守护可以使设计更持久，遗憾的是，大多数项目做得并不好。

除了上面这几点，还有一点就是，新的技术和框架会不断涌现，旧代码往往是不能有效使用这些新东西的。比如，Java 世界今天开发的主流是 Spring Boot，然而十年前，它还不存在。

虽然那时候已经有了 Spring，但那时候的主流开发方式还是打出一个 WAR 包，再部署到 Tomcat 上。所以，新出现的很多技术会提供更简单的做法，替换掉旧代码中笨拙的部分。

所以，到底怎么才能让自己的项目在设计上不断地演进，跟上时代发展的步伐，不断焕发新的活力呢？对于任何一个开发团队而言，这都是一个值得考虑的问题。

那么，这一讲，我们就来谈谈如何改进既有项目的设计。

从目标开始

在我的另外一个专栏《10x 程序员工作法》中，我讲过一个类似的主题，如何面对遗留系统。那里面的主要观点就是我们应该找到一个目标，然后小步改进，逐步向这个目标接近。

在那一讲中，我讲的重点主要在于改进的过程，而在这里，我打算从设计的角度再来审视一下这个问题。既然都是我的专栏，所以二者在解决问题上的思路一致的，都要先从找到目标开始。

大多数团队一说起改进，一般想的都是功能性方面的目标。比如，我原来的系统能支持100万的用户，现在要支持 1000 万的用户。这种改进固然是我们需要考虑的，甚至是迫不得已的。

但这种改进解决的是实现，因为不同量级的系统根本就不是一个系统，承载的用户量发生了变化，其实是一种需求的变化。但是，这种改变并不会让你的设计变好。

既然我们已经决定要改进了，就应该好好地把设计改进一下，而不只是把功能重新实现一遍。因为功能实现是你无论如何都必须做的，都是为别人做的，而设计的改进才是你为了自己做的，因为在未来的一段日子里维护这些代码的人是你。如果我们要做设计的改进，设定好改进设计的目标就显得尤为重要。

那设计改进的目标应该是什么呢？你可以先问一下自己这样一个问题，**如果有机会从头设计这个系统，它应**

该是什么样子呢？

这个问题可能会让很多程序员一下子愣住，因为他们每天都陷于忙碌的工作中，做的工作都是各种微调、各种打补丁，眼中只有一个具体微观的世界，却不曾有一个整体的思考。

是的，从头来过，它应该是什么样子。这是一个简单的问题，也是一个困难的问题。简单在于，它的字面意思很好理解。困难却在于，很多人一听到这个问题，直觉就要回避：

- 我的系统已经这么沉重了，怎么可能重来？
- 我有那么多的需求要做，哪有时间重做一遍？
- 我的系统那么复杂，重做一遍，出了问题谁来负责？

我承认，这些都是很现实的问题。但是，我的意思并不是让你真的一上来就动手，从零开始把系统重写一遍。这里的重点在于，**我们要找到改进的目标，也就是一个系统本来应有的面貌**。

这就是为什么我们前面要学习那么多设计一个系统的知识，否则，我们没有一个设计知识的沉淀，所谓的“重新设计”，弄不好我们就会回到原来的老路上去。

这时候，或许你突然想到一个严重的问题了，开启一次系统改进，如何处理人们的共识好像也是一件困难的事情，但这根本不是一个设计问题。想要真正地开启一次改进，就要让人们意识到，**设计一个系统和实施一次系统改进是两个完全不同的问题，可以分阶段地进行**。

我们只有把系统设计成它应有的样子，才算是确定了我们的目标。有了目标之后，接下来，我们才能制定改进路径，而把现有的系统一点一点从旧有的样子改动成新的样子，这是实施的过程。

好！我假设你已经搞定了周边人的共识，准备着手进行改进了。

改进的过程

现在你要重新设计这个系统了，或许你会想，这有什么难的？不就是照着原来的需求，重新来一遍吗？如果你真的还有原来的需求，能让你照着设计一遍。我真的只能说，你太幸运了。

在大部分真实的项目中，一个既有系统的情况是，没有人能够说出它到底承载了哪些需求。当然，主干部分是人人都知道的，但主干常常是九牛一毛，而更多的细节隐藏在代码中了。

一个长期存在的系统，开发者可能已经换了好几拨。了解当年那些需求的人可能早已不知所踪了，导致的结果就是，每一个工作在这个项目上的人都是只见树木不见森林。

在这种情况下，我们该怎么办呢？我给你**一个入手的起点，就是接口**。

在第4讲学习怎样理解一个系统的设计时，我们曾经说过，想要理解一个系统的设计，可以按照模型、接口和实现的这个框架去理解，其中，接口是模型能力的体现。

对于一个系统而言，接口也是使系统内部状态发生改变的原因，系统中的所有变化必然都是从某个接口开始的。既然没有人能够清楚地说明系统的现状，那么，我们从接口入手，了解系统的现状是一个非常现实的做法。毕竟，接口是不会骗人的。

不过，这里的接口不仅包括我们传统意义上的接口，也包括各种后台服务。前面我们讲了很多构建模型的内容，有了这个基础，我们再看后台服务，就会发现，后台服务只不过是按照某种规则触发模型的接口。比如，定时服务，就是定时地去调用模型的接口。所以，我们也要把这种接口梳理出来。

有了对于这些接口的了解，我们就对这个系统呈现哪些能力有一个认识了，就相当于获得了一份需求描述。基于这个认识， 我们来构建我们新的设计。

接下来，我们就要重新设计了，**这个改进设计的难点就是不要回到老路上**。我们需要按照一个正常设计的思路去走，该分离关注点的分离关注点，该重新组合的要重新组合。

之所以我要提示这一点，就是因为思维的惯性实在是太大了。比如说，在原有的系统内有一个叫订单的概念，我们就会习惯性地使用订单，而不是把商品订单、支付订单等概念分开。

一般而言，既有项目的设计有一个很大的问题就是各种信息混在一起，而能够把不同的信息拆分开来，对于设计而言，就是一个巨大的进步。

做好了新的设计，也就为我们后续的行动找到了新的方向。接下来，我们要做的是，对比新旧设计，找到一条改进路径。

永远不要指望一个真实的项目停下来，一步到位地进行改进。我们能够做的，唯有小心翼翼，一步一步向着目标前进。

对于不同的项目，选择的路径可能是不同的，有人会选择关键路径上的关键模块进行改进，也有人会选择影响较小的模块先进行探索，无论是哪种方案都是可以的。一个关键点就在于，动作要小。

学习过我的两个专栏的同学可能已经充分理解了我对小步前行的喜爱了。任何一个大动作，往往都意味着很长时间无法完成。在这个过程中，所有人都会提心吊胆。如果不能看到成果，很多人的信心都会随时间流失。所以，**在软件设计的改进过程中，积小胜为大胜才是一个合理的选项**。

还有一个关键点，要让所有相关利益人有一个共识。我又一次说到了共识，软件开发虽然是一个技术活，但归根结底还是一项团队活动，是一项人的活动。

既然涉及到诸多参与者，就一定要让大家形成一个共识。所以，系统改进，尤其是一个规模比较大的系统改进，一定要让所有人有共识。无论是开会也好，宣讲也罢，让大家对于改进的原因和改进的计划有个共同的预期是至关重要的。

更加具体的改进过程，我在《10x 程序员工作法》中有更细节的讨论，有兴趣的话，可以去参考一下。

虽然我在这里讲的是一个系统的改进过程，其实，同样的思路也可以运用在更小的模块中。只不过，更小模块意味着更少的接口、更低的复杂度以及更少的相关利益人。事实上，我反而鼓励你从小模块入手，一步到位去改进整个系统，难度系数是更大的，而小模块可以帮助你积累更多改进的经验，无论是设计，还是与人打交道。

总结时刻

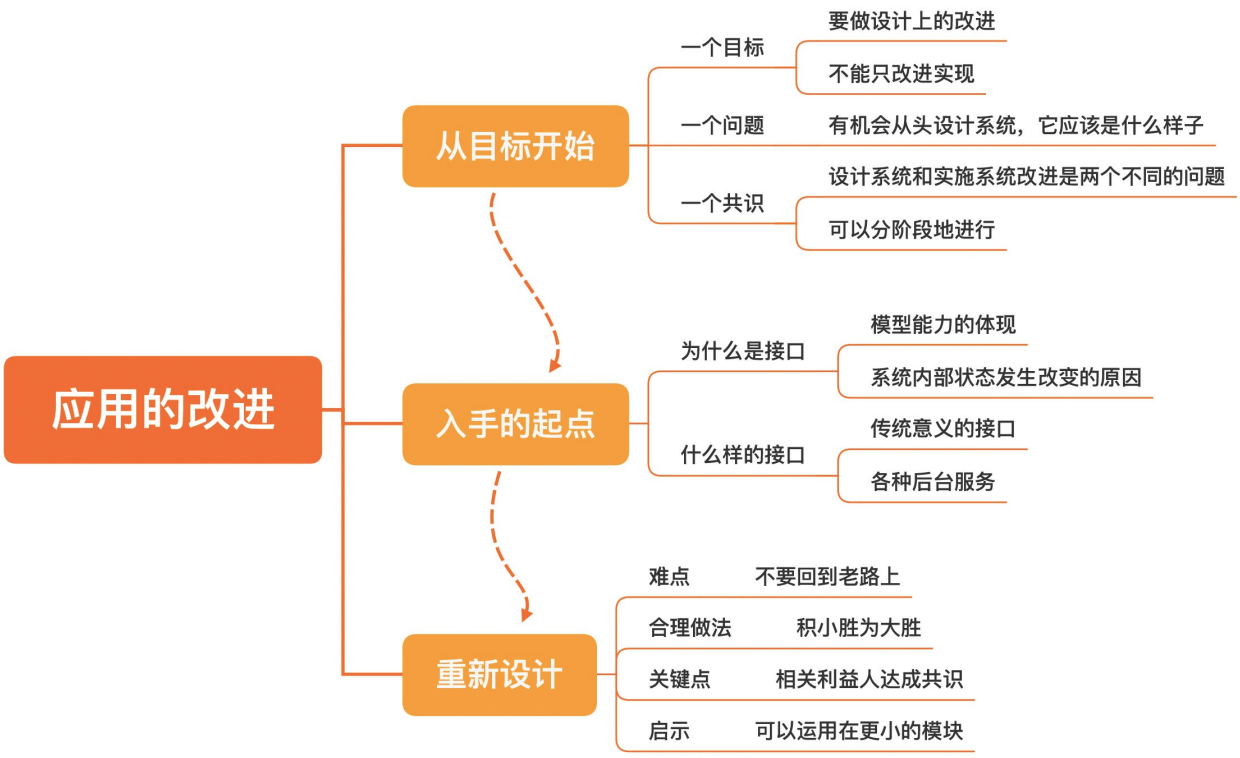
今天，我给你讲了如何改进一个既有软件的设计。一个软件放在时间长河中会有很多东西发生改变，即便是当初还算不错的设计，随着时间的累积，也可能积重难返。

改进一个软件的设计，首先，要确定改进的目标。改进的目标就是，重新设计这个软件，它应该设计成什么样子，让设计还原到它应有的本来面貌。寻找改进的起点，一部分可以从需求入手，还有一部分要从梳理接口入手。

设计改进的难点在于不要回到老路上，要做正常的设计，尤其是要把分解做好。

有了改进目标之后，接下来就是要找到一条改进路径，选择怎样的路径都是有道理的，但有两个关键点是非常重要的，一个是每步改进的动作要小；一个是要让相关利益人达成共识。

如果今天的内容你只能记住一件事，那请记住：**改进既有设计，从做一个正常的设计开始，小步向前**。



极客时间

思考题

最后，我想请你回想一下，你的系统在设计上存在着哪些问题，你打算怎么改进它呢？欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- 人间四月天 2020-08-10 11:36:25
  我们正在重构一个40多万行的应用，这个应用的开发，产品，业务都走光了，现在团队有的开发要重写，有的开发的意见是不动老代码，两种意见，我都是反对的，全部重写不现实，一个跑了5年的系统，需求根本没人弄的清楚，就算弄清楚需求，重写代码，就能做的更好？设计水平达到了吗？另外，不动老代码不现实，很多新需求，虽然是新需求，正因为设计的不好，不得不动老代码。面对这样的困境，我们还是采用小幅重构的办法，我们做了重构方案，目标清晰可落地，效果可检视。我们对重构进行分类。

1.瘦身，瘦身就是把系统的各种接口，页面接口，服务接口，job，通过监控统计已经下线的功能，把代码删除掉，最近一年没有访问的请求，把代码删除掉。另外，通过表，sql反向梳理，就是表里没有新数据，sql最近一年没有执行过，把这样的代码都删除掉。
2.重构，精准重构，1个是性能，梳理接口，统计超过一定阀值的接口，根据成本收益原则，确定优先级，分布实施，1个是功能复杂度，统计最近1年每个源代码文件提交的次数，我们的依据是开放封闭原则，一个类被频繁修改，就说明设计存在问题了。
3.新需求，新应用，对于业务领域是个全新的子域，我们坚决开发新应用解决，另外我们计划做服务标准化，对于新接口和服务，通过一个新应用实现。同样的遵守开放封闭原则。

我们已经重构了半年，下半年还要重构，我认为明年也要重构，重构得到领导的大力支持，我相信也是提升团队设计，开发水平的机会，加油吧！
[4赞]

作者回复2020-08-10 13:31:37
非常棒的分享，感谢你让大家看到实践者前行的足迹！

- 阳仔 2020-08-10 09:52:20
我非常赞同从改进一个小的模块设计开始，不断地小步前行的做法。
我最近也是在重构项目，虽然是APP项目，但这个项目很大，安装包已经有200M。
因为需求复杂，经历人员也比较多，结构设计也没有统一，有使用rxjava，dagger，MVP等等框架，交互层和业务层的逻辑交织在一起，改一个bug相当得耗时间。
现在第一步做法是梳理模块，统一每个模块对外的接口，这个改动目前来说是最小的，也是更进一步改造的基础。
曾多次问自己如果重新开始我会怎么设计这样的问题。其实回答这个问题可以帮你更好的理解现有系统的业务逻辑，也是你重构的目标，重新设计它是一个理想状态，不可能一下子就达到，但是可以不断地逼近目标。
[1赞]

作者回复2020-08-10 13:37:32
感谢分享！

- Jxin 2020-08-10 16:02:54
个人补充:
1.项目质量的好坏，与公司发展情况，项目价值息息相关。在业绩不好的条件下，举债前行就是合理的。所以我们讨论项目质量，谈设计和重构，都是建立在价值和公司业务发展需要的前提条件下的。(反之，逆势而为有违天和，事倍功半)

课后题:
1.问题: 最大的问题就是缺少共同设计。
2.解决思路: 开发与产品间的沟通应该基于统一、公开的业务建模。
3.问题描述: 产品和开发之间沟通都是基于一个个零散发散的功能点。在系统和业务间没有做一定的模型抽象。导致开发不了解业务全貌，产品不理解系统现状。进而在知识传承的成本就很大。往往新产品出的功能很难兼容现有系统（复用已有能力和兼容其他已有功能）;新的开发也很难看清系统的全貌。（进而问题就多，问题多团队氛围就比较差，就容易扯皮）
4.价值: 产品需要开发通过模型勾勒出系统的现状，屏蔽掉实现的复杂度。开发需要产品通过模型屏蔽掉无关的业务复杂性，串联整体业务脉络，快速准确的理解业务全貌。如此也有利于新人介入，知识传承。
6.外在问题: 因为不是一个人的事,所以天时地利很重要,同样的时间也可以做很多事情,没必要吃力不讨好不是。

旁外话
郑烨大佬的专栏都是金玉良言啊，不温不火着实费解。

作者回复2020-08-10 18:48:28
能够理解我写的内容都是吃过亏的人。

- Demon.Lee 2020-08-10 11:13:46
公司以前业务是2C，后来钱烧完了，开发人员走的差不多，现在做2B。
但把各个服务拆的非常细，几十个微服务，，几个人维护，现在也不敢做的大的改动。
想了想问题：
1. 微服务拆分的太多，各种服务之间RPC调用经常超时
2. 各个服务报错之后，没有事务和补偿机制，存在数据不一致问题
3. 微服务相关调用，一个服务挂了，好多功能停摆，无法用
4. 监控体系没有做，各种服务的状态和接口的调用情况不清晰
5. 服务熔断、链路追踪等没有

有时候，我在想，2B业务量不大，不如保持现在各个微服务的样子，只是不再一个个部署了，合并到一个进程里面跑算了（相当于模块化开发，把RPC调用禁掉）。只是有点回到老路的样子，大家有些不甘心。

作者回复2020-08-10 13:36:23
微服务应该是结果，而不应该成为目标。